浙江大学

本科实验报告

课程名称：计算机组成
姓名：曾一欣

学院：竺可桢学院

班级：求是科学班（计算机科学与技术）

专业：计算机科学与技术

学号：3180105144

指导老师：姜晓红
2020.6.30

浙江大学实验报告

课程名称： 计算机组成　　实验类型：综合

实验项目名称：Lab5 Microprogramming

学生姓名：曾一欣　　专业：计算机科学与技术　　学号： 3180105144

同组学生姓名： None　　指导老师： 姜晓红

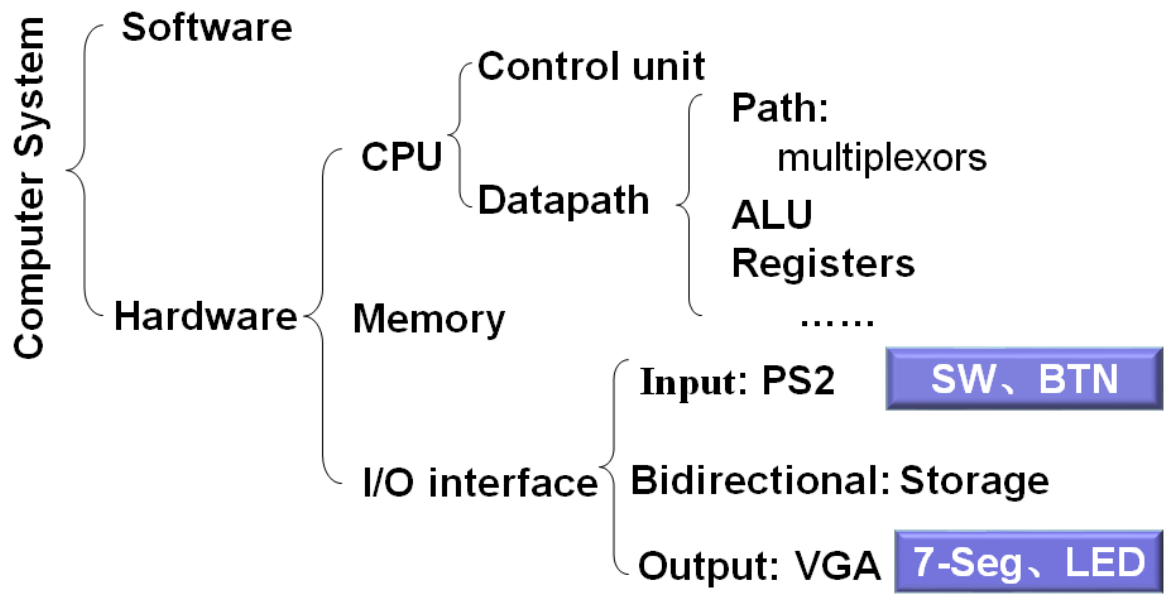
实验地点：无　　实验日期： 2020 年 6 月 30 日

一、 实验目的和要求

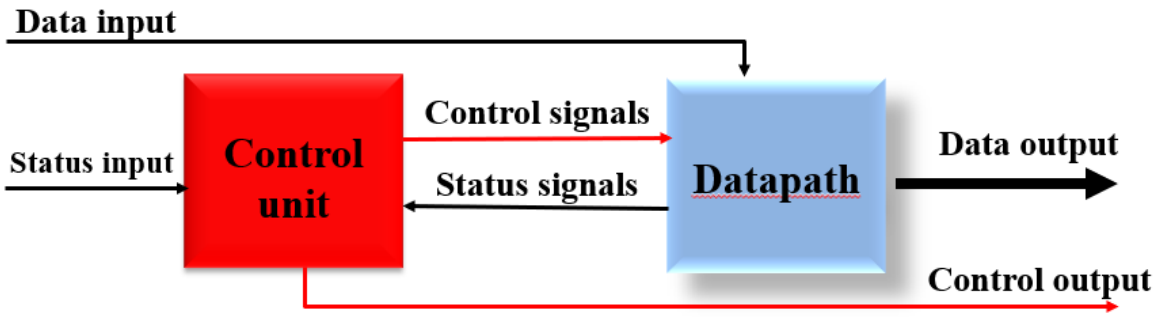
1. 设计microprogramming三个ROM内的真值表
2. 构建microprogramming基础下的控制部件
3. 利用上一次实验的数据通路，连接完成MCPU

二、 实验内容和原理

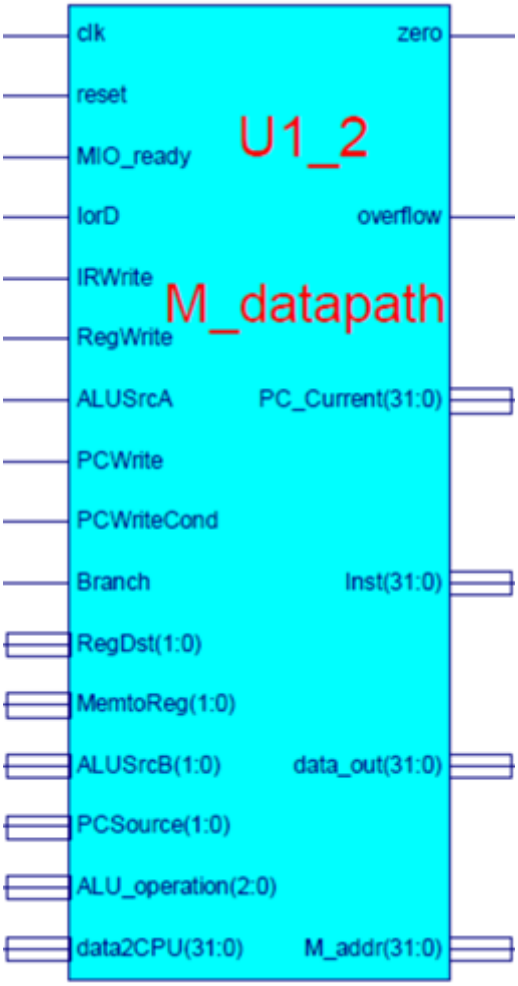
Decomposability of computer systems



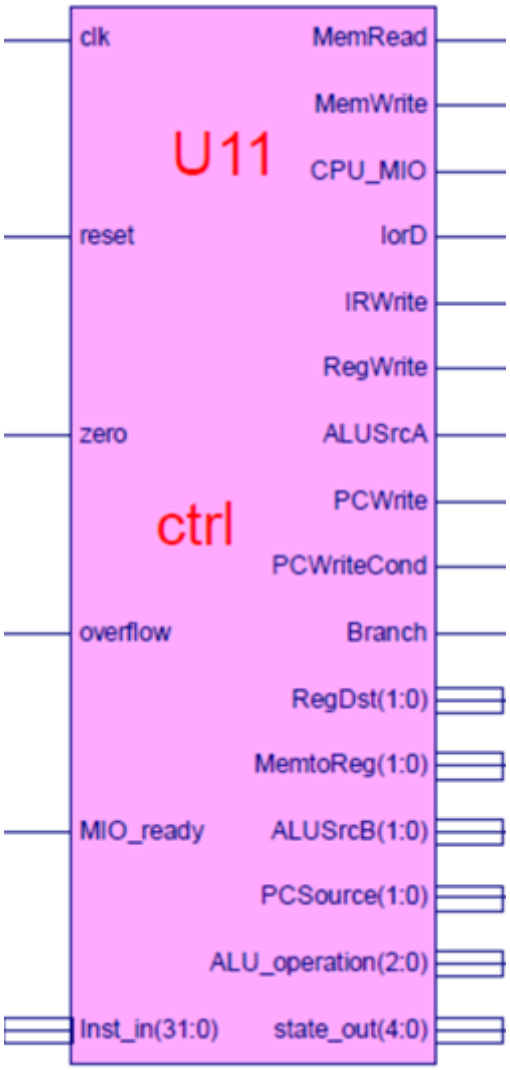
Digital circuit



DataPath



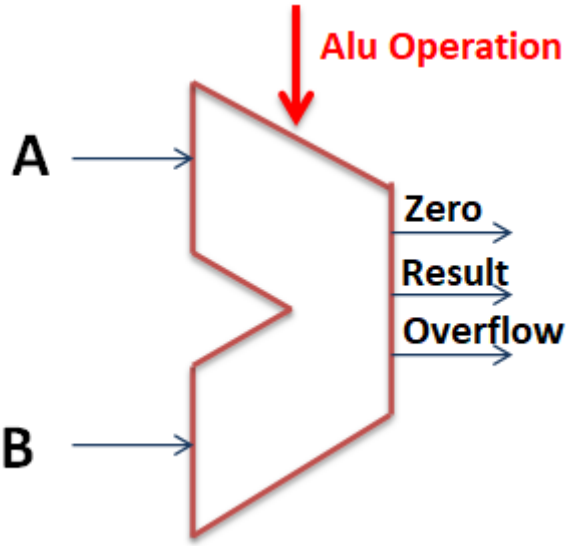
Controller



ALU

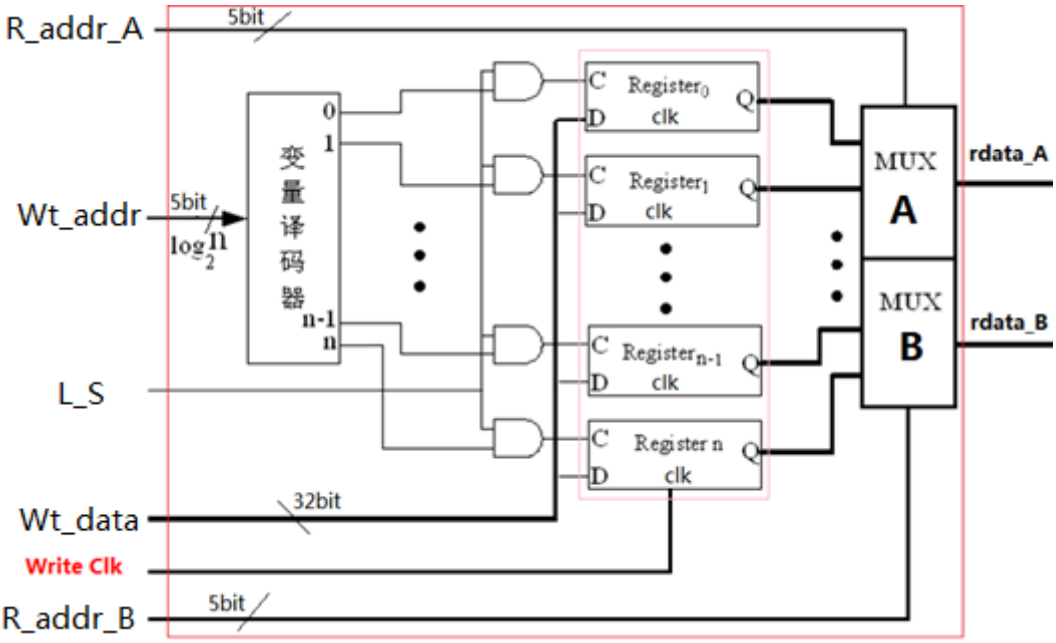
use the same module ALU in SCPU

ALU Control Lines	Function	note
000	And	兼容
001	Or	兼容
010	Add	兼容
110	Sub	兼容
111	Set on less than	
100	nor	扩展
101	srl	扩展
011	xor	扩展



Register file

use the same module in SCPU



State table

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Truth table

ALU	0	0	0	0	0	0	1	0	0	0	ALUOp1
	0	0	0	0	0	0	0	0	1	0	ALUOp0
SRC1	0	0	1	0	0	0	1	0	1	0	ALUSrcA
SRC2	0	1	1	0	0	0	0	0	0	0	ALUSrcB1
	1	1	0	0	0	0	0	0	0	0	ALUSrcB0
Register control	1	0	0	0	0	0	0	0	0	0	IRWrite
	0	0	0	0	1	0	0	1	0	0	RegWrite
	0	0	0	0	0	0	0	1	0	0	RegDst
	0	0	0	0	1	0	0	0	0	0	MemtoReg
Memory control	1	0	0	1	0	0	0	0	0	0	MemRead
	0	0	0	0	0	1	0	0	0	0	MemRead
	0	0	0	1	0	1	0	0	0	0	IorD
PCWite	0	0	0	0	0	0	0	0	0	1	PCSource1
	0	0	0	0	0	0	0	0	1	0	PCSource0
	1	0	0	0	0	0	0	0	0	1	PCWrite
	0	0	0	0	0	0	0	0	1	0	PCWriteCond
Seq	1	0	1	1	0	0	1	0	0	0	Addrctl1
	1	1	0	1	0	0	1	0	0	0	Addrctl0

三、 实验过程和数据记录及结果分析

Design the whole truth table and state table

the table in the slides is not complete, so I design the complete tables following the way in the power points.

Microprogramming 取R I-type LUI Bne Jump 细节 SLL SRL

address	label	ALUcontrol	SRC1	SRC2	Registercontrol	Memory	PCwrite	Sequencing
0000	Fetch	Add	PC	4		ReadPC	ALU	Seq
0001		Add	PC	Extshft	Read			Dispatch1
0010	Mem1	Add	A	Ext				Dispatch2
0011	LW2					ReadALU		Seq
0100					Write MDR			Fetch
0101	SW2					Write ALU		Fetch
0110	Rformat1	Func Code	A	B				Seq
0111					WriteALU			Fetch
1000	BEQ1	Subt	A	B			ALUout.Cond	Fetch
1001	JUMP1 (J)						Jump address	Fetch

微指令编码

ALU { 0 0 0 0 0 0 1 0 0 0 / 0 0 0 0 0 0 0 0 1 0

SRC1 0 0 1 0 0 0 1 0 1 0

SRC2 { 0 1 1 0 0 0 0 0 0 0 / 1 1 0 0 0 0 0 0 0 0

Reg Control { 1 0 0 0 0 0 0 0 0 0 IRwrite / 0 0 0 0 1 0 0 1 0 0 RegWrite / 0 0 0 0 0 0 0 1 0 0 RegDst → 2位 / 0 0 0 0 1 0 0 0 0 0 MemtoReg → 扩展为地址

Mem Control { 1 0 0 1 0 0 0 0 0 0 MemRead / 0 0 0 0 0 1 0 0 0 0 MemWrite / 0 0 0 1 0 1 0 0 0 0 IorD

PC write { 0 0 0 0 0 0 0 0 0 1 / 0 0 0 0 0 0 0 0 1 0 } PCSource / 1 0 0 0 0 0 0 0 0 1 PCWrite / 0 0 0 0 0 0 0 0 1 0 PCWriteCond

seq { 1 0 1 1 0 0 1 0 0 0 / 1 1 0 1 0 0 1 0 0 0 → 00: Fetch 01: dispatch1 11: Seq 10: dispatch2

1 0 0 1 0 1 0 0 0 0 MemIO

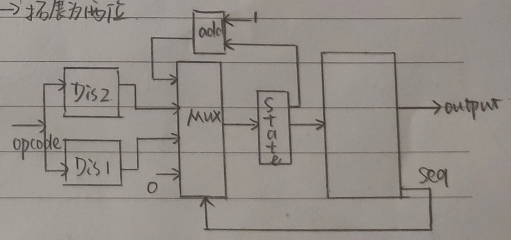
0 0 0 0 0 0 0 0 1 0 branch

Dispatch1

op		value	
000000	R-format	0110	000000 / 001101 / 001010 → 1010 (I)
000010	jmp	1001	
000100	beq	1000	
100011	lw	0010	000111 → 1100 (LUI)
101011	sw	0010	000101 → 1101 (Bne)

Dispatch2

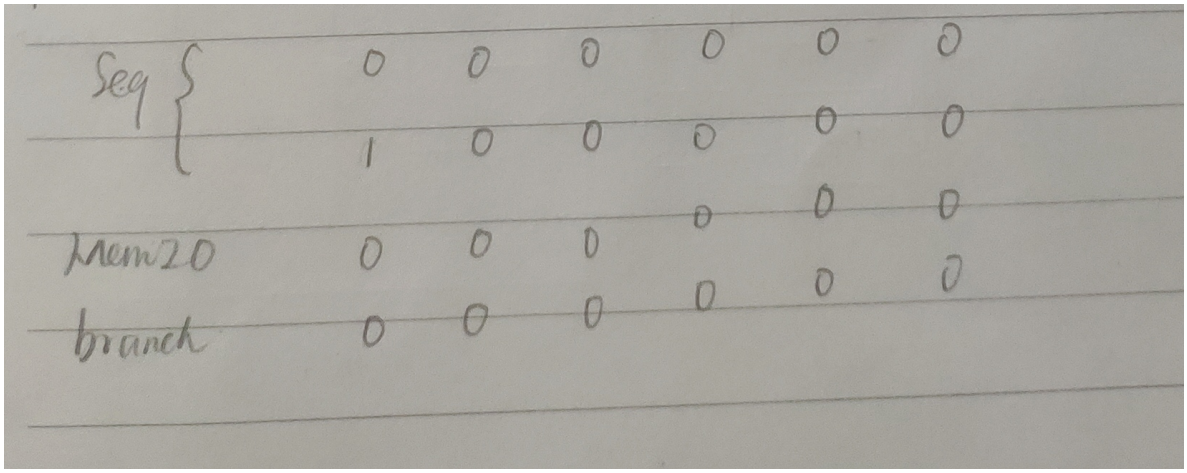
op		value
100011	lw	0011
101011	sw	0101



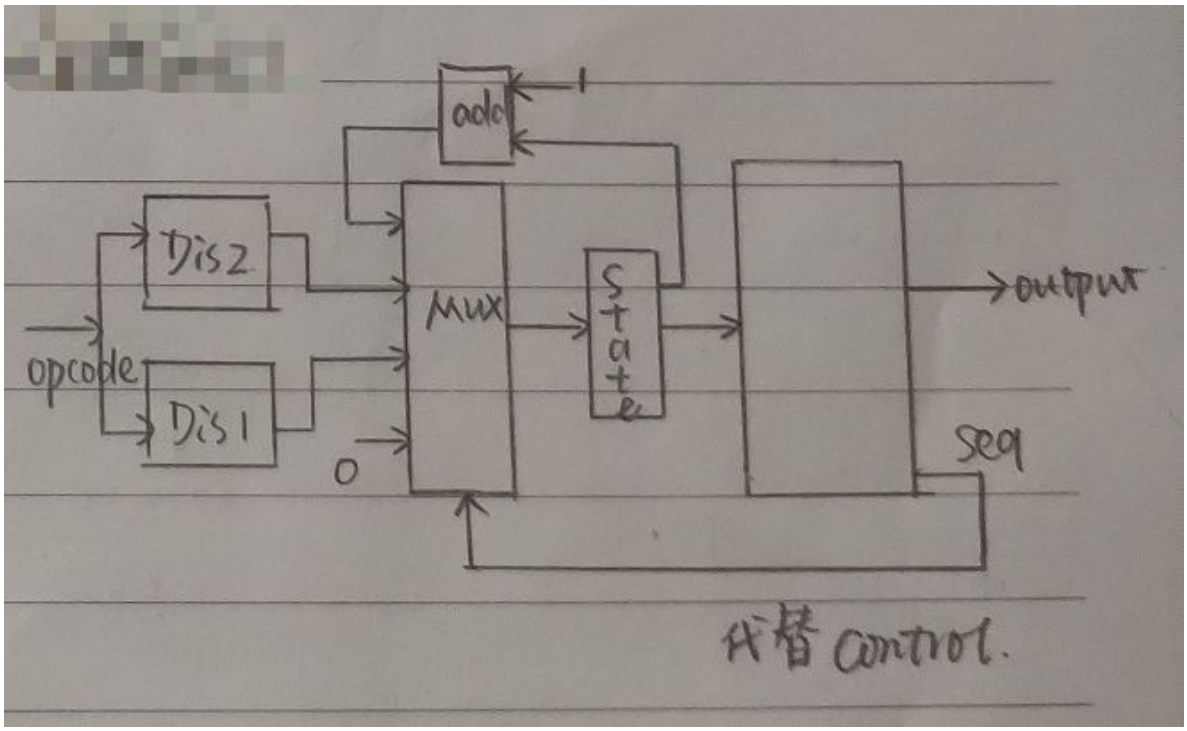
状态 control.

address	label	label (befor)	Sequencing
1010	I-type	I-Exc	Seq
1011		I_WB	Fetch
1100	LUI	Lui	Fetch
1101	Bne	Bne_Exc	Fetch
1110	Jr	Jr	Fetch
1111	Jal	Jal	Fetch

ALU	1	0	0	0	0	0
	1	0	0	1	0	0
SrcA	1	1	0	1	1	0
SrcB	1	1	1	0	0	1
	0	0	1	0	0	1
IRWrite	0	0	0	0	0	0
RegWrite	0	1	1	0	0	1
RegDst	0	0	0	0	0	1
	0	0	0	0	0	0
MemtoReg	0	0	1	0	0	1
	0	0	0	0	0	1
MemRead	0	0	0	0	0	0
MemWrite	0	0	0	0	0	0
IorD	0	0	0	0	0	0
PCSrc	0	0	0	0	0	1
	0	0	0	1	0	0
PCWrite	0	0	0	0	1	1
PCWriteCond	0	0	0	1	0	0



Design the ctrl circuit



Modules

ROMS

dispatch1

```
memory_initialization_radix=2;
memory_initialization_vector=
0110,0000,1001,0000,1000,1100,0000,0000,
1010,0000,1010,0000,0000,1010,0000,1100,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0010,0000,0000,0000,0000,
0000,0000,0000,0010,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000;
```

dispatch2

```
memory_initialization_radix=2;
memory_initialization_vector=
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0011,0000,0000,0000,0000,
0000,0000,0000,0101,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000,
0000,0000,0000,0000,0000,0000,0000,0000;
```

microprogramming

```
memory_initialization_radix=2;
memory_initialization_vector=
00001100000010000101110,
00011000000000000000100,
00110000000000000001000,
00000000000010100001110,
00000010001000000000000,
00000000000001100000010,
10100000000000000001100,
00000010100000000000000,
01100000000000001010001,
00000000000000010100000,
11110000000000000000100,
00110010000000000000000,
00011010010000000000000,
01100000000000001010000,
00100000000000000100000,
00011011011000010100000;
```

State register

```
module state(
input clk,
input [3:0]statein,
output [3:0]state
    );
reg [3:0]s;
always @(posedge clk)begin
    s<=statein;
end
assign state=s;

endmodule
```

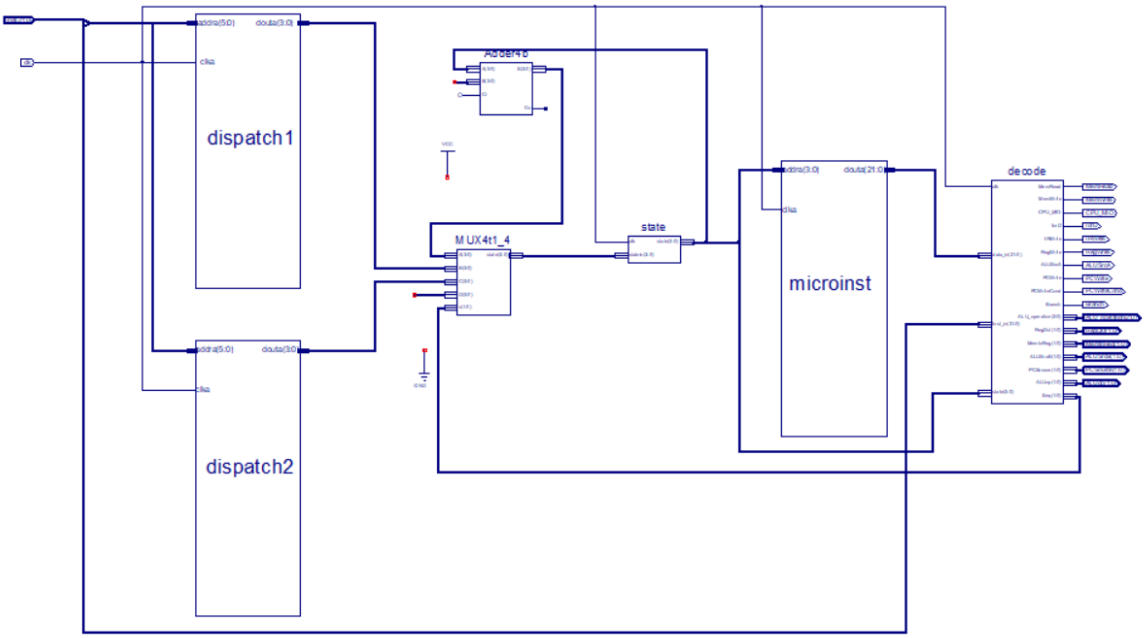
decoder

```
module decode(
                input [21:0]data_in,
                input [31:0] Inst_in,
                input [3:0]state,
                input clk,
                output MemRead,
```

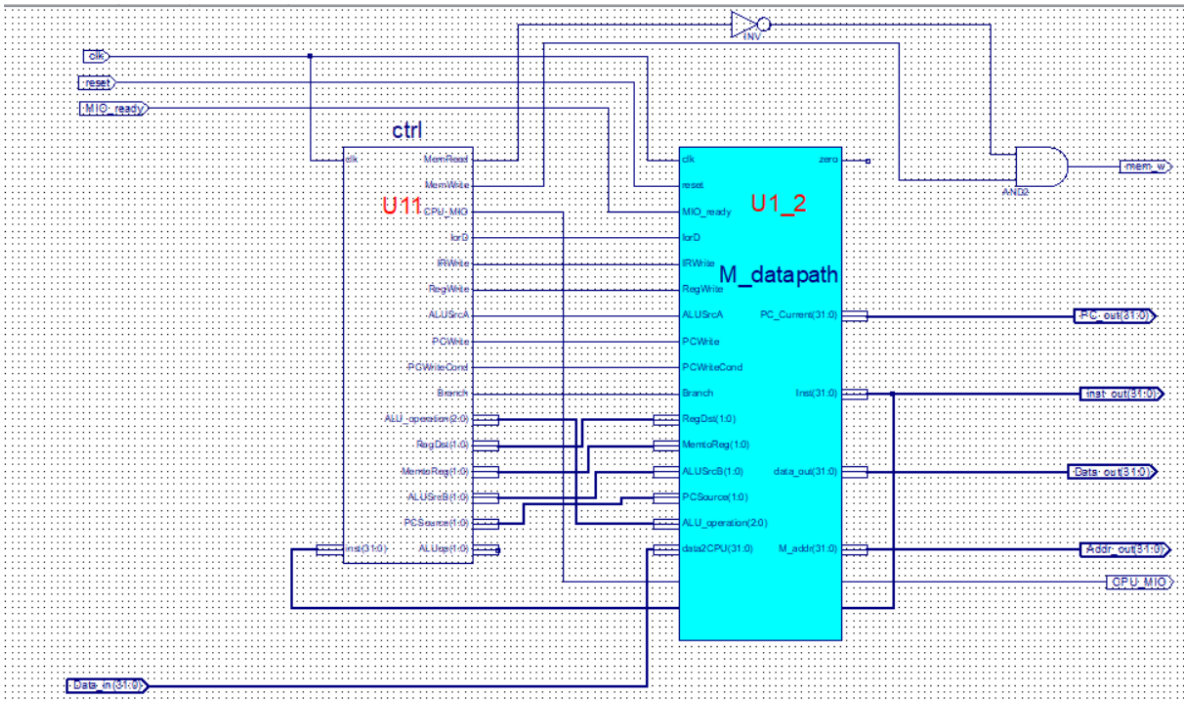
```
                output MemWrite,
                output reg[2:0]ALU_operation,

                output CPU_MIO,
                output IorD,
                output IRWrite,
                output [1:0]RegDst,
                output RegWrite,
                output [1:0]MemtoReg,
                output ALUSrcA,
                output [1:0]ALUSrcB,
                output [1:0]PCSource,
                output PCWrite,
                output PCWriteCond,
                output Branch,
                output [1:0]ALUop,
                output [1:0]Seq
    );

assign ALUop=data_in[21:20];
assign ALUSrcA=data_in[19];
assign ALUSrcB=data_in[18:17];
assign IRWrite=data_in[16];
assign RegWrite=data_in[15];
assign RegDst=data_in[14:13];
assign MemtoReg=data_in[12:11];
assign MemRead=data_in[10];
assign MemWrite=data_in[9];
assign IorD=data_in[8];
assign PCSource=data_in[7:6];
assign PCWrite=data_in[5];
assign PCWriteCond=data_in[4];
assign Seq=data_in[3:2];
assign CPU_MIO=data_in[1];
assign branch=data_in[0];

parameter
AND=3'b000,OR=3'b001,ADD=3'b010,SUB=3'b110,NOR=3'b100,SLT=3'b111,XOR=3'b011,SRL=
3'b101,SLL=3'b110;


always @ * begin
    case(state)
    4'b0110:begin
          case (Inst_in[5:0]) //R-type OP
            6'b100010: ALU_operation <= SUB;
            6'b100100: ALU_operation <= AND;
            6'b100101: ALU_operation <= OR;
            6'b100111: ALU_operation <= NOR;
            6'b101010: ALU_operation <= SLT;
            6'b000010: ALU_operation <= SRL; //SP3 shfit 1bit right
            6'b000000: ALU_operation <= SLL;
            default: ALU_operation <= ADD;
            endcase
        end
    4'b0111:begin
              case (Inst_in[5:0]) //R-type OP
            6'b100010: ALU_operation <= SUB;
```

```
            6'b100100: ALU_operation <= AND;
            6'b100101: ALU_operation <= OR;
            6'b100111: ALU_operation <= NOR;
            6'b101010: ALU_operation <= SLT;
            6'b000010: ALU_operation <= SRL; //SP3 shfit 1bit right
            6'b000000: ALU_operation <= SLL;
            default: ALU_operation <= ADD;
            endcase
        end
    4'b1010:begin
        case (Inst_in[31:26])
            6'b000000: case(Inst_in[5:0])
                                6'b000000: ALU_operation = SLL;
                                6'b000010:ALU_operation=SRL;
                            endcase
            6'b001000: ALU_operation=ADD;
            6'b001101: ALU_operation=OR;
            6'b001010: ALU_operation=SLT;
            default: ALU_operation <= ADD;
            endcase
        end
    4'b1011:begin
        case (Inst_in[31:26])
            6'b000000: case(Inst_in[5:0])
                                6'b000000: ALU_operation = SLL;
                                6'b000010:ALU_operation=SRL;
                            endcase
            6'b001000: ALU_operation=ADD;
            6'b001101: ALU_operation=OR;
            6'b001010: ALU_operation=SLT;
            default: ALU_operation <= ADD;
            endcase
        end
    default :ALU_operation=ADD;
    endcase
end

endmodule
```

Ctrl



Top



四、 讨论与心得

In this experiment I tried to construct a Multi-cycle CPU using micro-programming.

Because there's IPcore in the experiment, I can't do some simulation to verify the correctness of the whole system. However, as the theory of micro-programming is actually quiet simple if it is totally understood, I believe if the truth table and state graph is construct carefully, there won't be too much mistakes.

As it's the last experiment of the course, I'd like to talk about my idea about the whole experiments. It's really a pity that we have to stay at home and do all the experiment. There's no procedure about debug and rethought about the experiment, which makes writting the "discussion and thoughts" part really difficult. What's more, when doing the next experiment, there's usually necessarity to use the result of the last experiment. As we can't prove the correctness of the previous experiment, this procedure cause a lot of difficulties. However,

despite all these unhappiness, by doing the experiments we complete a simple CPU all by ourselves which is really interesting. And I think it's one of the most significant part of the courses.